2. Procesoare (UCP)

În figura 2.1 este prezentata organizarea unui calculator simplu, organizat în jurul unei magistrale.

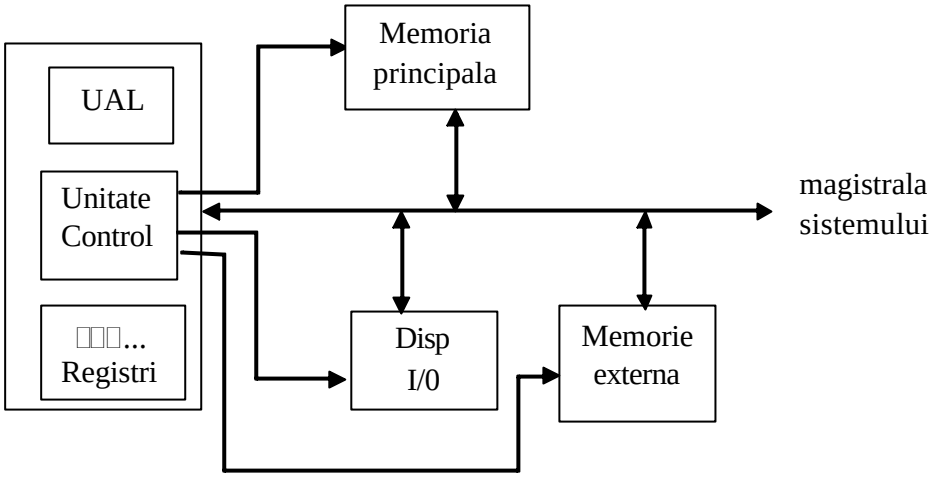

Figura 2.1.

Componentele sunt conectate printr-o magistrala; aceasta este formata dintr-o multine de fire paralele pe care sunt transmise adrese, date si semnale de control. Magistralele se pot afla în exteriorul UCP conectând o memorie si disp I/0, dar si în interiorul UCP, dupa se va vedea în curând.

UCP este alcatuit din mai multe parti distincte. Unitatea de control raspundede de extragerea instructiunilor din memoria principala si executarea lor. UAL executa operatiile necesare îndeplinirii instructiunilor.

UCP mai contine si o memorie redusa ca dimensiune, foarte rapida, plasata pentru depozitarea rezultatelor temporare si a anumitor informatii de control. Aceasta e formata dintr-un numar de registri. De obicei toate registrele au aceasi dimensiune. Registrele pot fi citite si scrise cu mare viteza deoarece se afla în interiorul UCP. Cel mai important registru este PC (program counter) care indica instructiunea urmatoare ce va fi extrasa pentru executie. Important este si registrul IP – registru de instructiuni (instruction register), în care se pastreaza instructiunea în curs de executie. Mai exista si alte registre, de uz general si altele pentru scopuri bine determinate.

2.1. Organizarea UCP

Organizarea interna a unei parti dintr-o UCP von Neumann tipica este data mai în detaliu în Fig. 2.2.

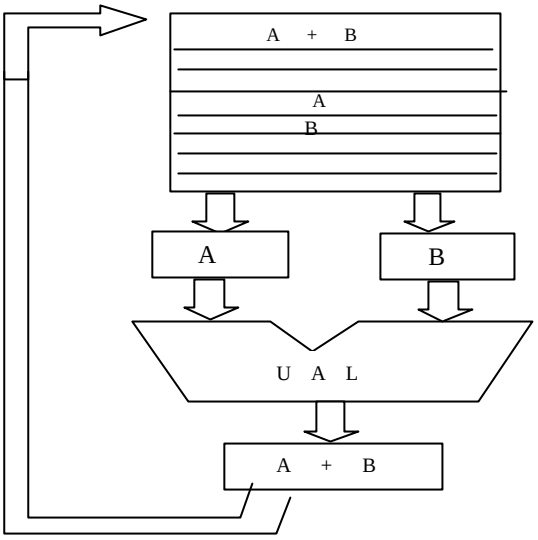

Fig. 2.2. Calea de date a unei masini von Neumann tipice

Aceasta parte se numeste *calea de date* si include registri (între 1 si 32), UAL si mai multe magistrale de legatura. Registrele trimit date catre cele 2 registre de intrare în UAL, notate A si B. Acestea pastreaza datele de intrare ale UAL în timp ce acesta calculeaza. UAL executa asupra datelor operatii simple (adunari , scaderi, functii logice) al caror rezultat este depus ulterior tot în registri prin intermediul registrului de iesire. Ulterior registrul poate fi scris (adica depozitat) în memorie. În exemplu este ilustrata adunarea.

Majoritatea instructiunilor fac parte din categoria registru–memorie sau registru–registru. Instructiunile registru-memorie permit cuvintelor din memorie sa fie încarcate în registri, unde pot fi folosite ca date de intrare pentru UAL, de exemplu. Altele pentru depozitarea în memorie.

(“Cuvintele” sunt unitati de date transferate între memorie si registru. Un cuvânt are lungimea variabila, în functie de calculatorul de care discutam. Se va detalia subiectul la organizarea memoriei).

Instructiuni registru–registru. O astfel de instructiune extrage doi operatori din registru, îi aduce în registrele de intrare ale UAL, executa o operatie asupra lor, si depune rezultatul din nou în registru. Procesul prin care cei doi operatori sunt trecuti prin UAL si rezultatul este depozitat se numeste ciclu al caii de date (data path cycle) si este inima celor mai multe UCP. El defineste în mare masura ce poate face masina. Cu cât ciclul caii de date e mai rapid, cu atât masina merge mai repede.

2.2. Executia unei instructiuni.

UCP executa instructiunea printr-o scriere de pasi, care sunt urmatorii:

1. Transfera instructiunea urmatoare din memoria IR;
2. Schimba controlul de astfel încât sa indice urmatoarea instructiune;
3. Determina tipul instructiunii extrase;
4. Daca instructiunea are nevoie de un cuvânt de memorie determina unde se gaseste acesta;

5. Extrage cuvântul în unul din registrele UCP;
6. Executa instructiunea;
7. Reia de la pasul 1 pentru a începe executia instructiunilor urmatoare.

Deseori aceasta secventa de pasi este denumita ciclu extrage – decodifica – executa (fetch–decode–execute).

În A. Tanenbaum este data aceasta secventa ca o procedura Java cu numele **interpret.**

Însusi faptul ca este posibil sa se scrie un program care sa imite functionarea UCP demonstreaza ca un program nu trebuie executat de o UCP hardware, adica de o cutie plina cu componente electronice. În loc de aceasta, un program poate fi executat de un alt program care sa-i extraga, examineze si execute instructiunile. Un program care extrage, examineaza si executa instructiunile altui program se numeste interpretor (asa cum s-a mentionat deja).

Lasfârsitulanilor '80, IBM a recunoscut avantajele multiple ale dezvoltarii unei singure familii de masini care sa execute acelasi instructiuni. IBM a introdus termenul de arhitectura pentru a descrie aceasta compatibilitate. O noua familie de calculatoare are deci aceeasi arhitectura, dar multe implementari diferite, capabile sa execute acelasi program, diferente fiind doar la capitolele viteza si pret.

Cum poate fi constituit un calculator ieftin care sa poata executa toate instructiunile complexe ale calculatoarelor scumpe? Raspunsul sta în interpretare. Aceasta tehnica a fost sugerata de Wilkes (1951). Rezultatul a fost masina IBM/360, o famile de calculatoare compatibile. O implementare hardware directa a fost folosita doar pentru modelele cele mai scumpe.

Calculatoarele simple cu instructiuni interpretate mai aveau si alte avantaje:

1. Posibilitatea de a corecta pe bc instructiunile incorect implementate, sau chiar de a rezolva deficiente de proiectare ale hardware-ului de baza.
2. Posibiliatatea de a adauga noi instructiuni la un cost minim.
3. Proiectarea structurata, ce permite dezvoltarea, testarea si documentarea eficienta.

Datorita cererii crescute de calculatoare în anii `70 si cresterea rapida a capacitatilor de calcul, a fost favorizata proiectarea calculatoarelor ce folosesc interpretoare. Posibilitatea de a ajusta hardware-ul si interpretorul a devenit un mod de proiectare foarte eficient pentru procesoare ieftine.

Acest current a ajuns la apogeu cu calculatorul VAX al DEC, care dispunea de câteva sute de instructiuni si peste 200 de moduri diferite de specificare a operanzilor pentru fiecare instructiune. Aceasta abordare a dus la includerea a multe instructiuni cu utilitate marginala, ce erau greu de executat direct.

Cu toate ca microprocesoarele au început pe 8 biti si cu seturi de instructiuni simple, spre sfârsitul anilor `70 chiar si microprocesoarele ajunsesera sa fie proiectate pe baza interpretoarelor.

Succesul microprocesoarelor Motorola 68000, care avea un set mare de instructiuni interpretate si esecul care concomitant al microprocesoarelor Z8000, care avea tot un set mare de instructiuni, dar fara un interpretor, a demonstrate avantajele interpretorului în lansarea rapida a unui nou microprocesoar.

Un alt factor favorizant al dezvoltarii interpretarii în acea epoca a fost existenta memoriilor rapide numai pentru citire, numite memorii de control. (control stores), memorii în care erau depozitate interpretoarele. Sa presupunem ca o instructiune interpretata tipica la M68000 necesita 10 instructiuni, numite microinstructiuni, fiecare durând 100µs, si 2 adesari la memorie, fiecare durând 500 µs $\Rightarrow$ timpul total 2000 µs, de doua ori mai mult decât cel mai bun timp posibil obtinut prin executia directa a instructiunilor.

2.3. RISC si CISC

La sfârsitul anilor `70 s-au facut experimente cu instructiuni foarte complexe, posibile datorita interpretorului. Proiectantii au încercat sa acopere "interstitiul semantic" (semantic gap) dintre posibilitatile masinilor si limbajele de programare de nivel înalt. Toata lumea încerca sa proiecteze masini mai complicate (cu un numar mai mare de instructiuni mai puternice), asa cum astazi se cauta proiectarea unor sisteme de operare, retele, procesoare de text mai puternice.

Au existat 2 exceptii. John Cocke la de IBM a încercat sa incorporeze o parte din ideile lui Seymour Cray într-un minicalculator de înalta performanta. A rezultat modelul experimental 801. În 1980 un grup de la Berkeley, condus de David Patterson si Carlo Sequin a proiectat cipuri VLSI pentru un UCP fara interpretor. Ei au introdus termenul RISC pentru acest concept (Reduced Instruction Set Computer). Putin mai târziu, în 1981, la Stanford, John Henessy a proiectat si fabricat un cip întrucâtva diferit, pe care l-a numit MIPS. Aceste cipuri au evoluat în produse comerciale de notorietate, procesoarele SPARC si respective MIPS.

Aceste procesoare erau mult diferite fata de cele existente în acel moment. Deoarece nu trebuiau sa pastreze vreo compatibilitate cu produsele anterioare, proiectantii au ales seturi de instructiuni care sa maximizeze performanta sistemului. Daca la început accentual cadea pe instructiuni care se execute rapid, în curând s-a constatat ca proiectarea unor instructiuni care sa poata fi lansate (issued=pornite) rapid este cheia succesului. Cât timp dura în total o instructiune conta mai putin decât numarul de instructiuni ce puteau fi lansate într-o secunda.

Procesoarele la care ne referim erau caracterizate de un set de instructiuni mic (aproape 50) fata de 200–300 la DEC VAX sau marile calculatoare IBM.

De aici acronimul RISC, în contract CISC ce înseamna Complex Instuction Set Computer. Suporterii RISC sustineau ca modul cel mai bun de a proiecta un calculator este sa ai un set de instructiuni simple care sa se execute într-un singur ciclu al caii de date (Fig. 2.2). Argumentul era viteza de executie: chiar daca masina RISC va face în 4 -5 instructiuni ceea ce va face o masina CISC într-o instructiune, daca instructiunile RISC sunt de 10 ori mai rapide, arhitectura RISC va fi în câstig. În acel moment si viteza memoriilor principale crescuse, ajungând din urma viteza memoriilor de control, fapt ce a favorizat si el puternic masinile RISC.

S-ar putea crede ca, date fiind performantele oferite de arhitectura RISC, masinile RISC (cum ar fi DEC Alpha) vor detrona masinile CISC (cum ar fi Intel Pentium) de pe piata. De ce nu s-a întâmplat asa?

În primul rand din cauza compatibilitatii cu modele anterioare si a banilor investiti în software pentru gama Intel. În al doilea rand, Intel a reusit sa incorporeze aceleasi idei într-o arhitectura CISC. Începând cu 486, UCP-urile Intel contin un nucleu RISC ce executa instructiunile mai simple (si în general mai des întâlnite) într-un singur ciclu al caii de date, interpretând instructiunile mai complicate în modul caracteristic CISC. Rezultatul net este ca instructiunile uzuale sunt rapide si instructiunile exotice mai lente. Chiar daca aceasta abordare nu e la fel de rapida ca cea RISC pura, ea permite obtinerea unei performante globale competitive.

2.4. Principii de proiectare pentru calculatoarele moderne

La 20 de ani de la aparitia masinilor RISC, anumite principii au fost acceptate ca fiind un mod bun de proiectare în conditiile tehnologiei hardware actuale.

Daca intervine o schimbare brusca a tehnologiei (de exemplu printr-un nou process de fabricatie ciclul de memorie devine de 10 ori mai rapid decât ciclul UCP) aceste principii vor trebui revizuite. Exista un set de principii de proiectare (numite principiile proiectarii RISC) pe care arhitectii UCP încearca sa le urmeze, dar datorita constrângerilor externe, cum ar fi cerinta copatibilitatii cu arhitecturi existente, aceste principii sunt obiective ce sunt încercate a fi atinse:

a) – Toate constructiile executate de catre hardware (adica nu sunt interpretate prin microinstructiuni). Daca nu e posibil, pe calculatoarele CISC, instructiunile mai complicate pot fi sparte în parti separate, ce pot fii apoi executate ca o secventa de microinstructiuni.

b) – Maximizeaza rata de lansare în executie a instructiunilor. Daca pot fi lansate simultan mai multe instructiuni simultan creste corespunzator viteza $\Rightarrow$ importanta paralelismului (control–flow parallelism). Instructiunile nu sunt lansate întotdeauna în ordinea din program. Daca o instructiune scrie într-un registru, iar asemanatoarea citeste acel registru, este nevoie de atentie pentru a ne asigura ca instructiunea a doua citeste registrul dupa ce el a fost înscris corect de prima. Sunt necesare unele artificii, dar exista potential pentru cresterea performantei prin executarea simultana a mai multor instructiuni.

c) – Instructiunile trebuie sa fie usor de decodificat $\Rightarrow$ necesitatea unui format fix al instructiunilor, cu un numar mic de câmpuri si cu dimensiune prestabilita.

d) – Numai instructiunile LOAD si STORE trebuie sa acceseze memoria. Accesul la memorie poate dura mult, iar întârzierile sunt imprevizibile, deci ar fi bine sa se poata suprapune executia acestor instructiuni cu a altora, ce lucreaza cu registrele.

e) Poseda registre suficiente (cel putin 32).

Din cele spuse, rezulta ca este utila implementarea paralelismului la nivelul UCP. Exista în principal 2 tipuri de parallelism:

- control-flow paralelism (executarea simultana a mai multor instructiuni)
- data parallelism (executarea aceleasi instructiuni cu date împartite pe procesoare).

2.5. Parallelism la nivelul instructiunii

În acest caz paralelismul este exploatat în cadrul instructiunilor individuale, pentru a face masina sa lanseze în executie mai multe instructiuni/secunda.

Pipeline (banda de ansamblare sau conducta) .

Concepul a existat din `60 la masini IBM prin citirea în avans a unei instructiuni din memorie si pastrarea ei într-un set de registri numit prefetch buffer. Citirea în avans împarte instructiunile în doua parti: extragerea si executarea propriu-zisa. Conceptul pipeline extinde strategia. Instructiunea e împartita în mai multe parti, de fiecare parte ocupându-se o componenta hardware, toate aceste componente hardware putând sa functioneze în paralel. Fig. 2.3 ilustreaza o banda de asamblare cu 5 unitati numite si stages (segmente, etape). Segmentul 1 extrage instructiunea din memorie si o plaseaza într-un registru tampon. Segmentul 2 o decodifica, determinându-i tipul si operanzii. Segmentul 3 localizeaza si extrage operanzii, fie din registri, fie din memorie. Segmentul 4 executa instructiunea, de obicei rulând operanzii prin calea de date, iar segmentul 5 scrie rezultatul în registri.

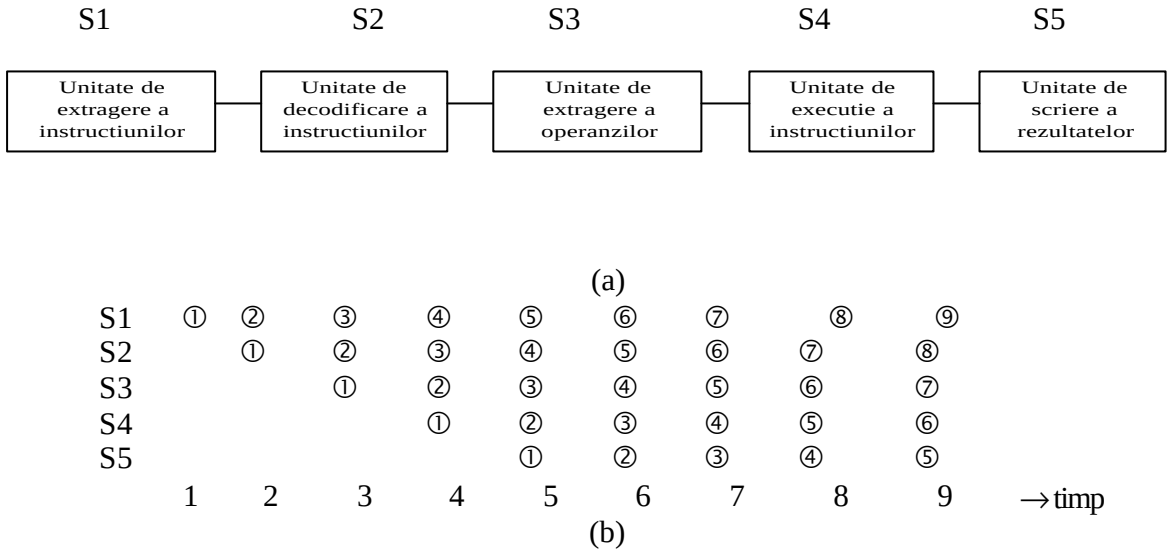

Fig. 2.3. O banda de asamblare de 5 segmente (a)
Starea fiecarui segment în functie de timp (b)

În figura 2.3 –b vedem cum opereaza o banda de asamblare în functie de timp. În ciclul 1, segmentul S1 lucreaza asupra instructiunii 1 (o extragere din memorie). În ciclul 2, S2 decodifica instructiunea 1. Tot în ciclul 2, S1 extrage instructiunea 2. În ciclul 3, S3 extrage operanzii pentru instructiunea 1, S2 decodifica instructiunea 2 si S1 extrage instructiunea 3.

Daca un ciclu masina dureaza x ns, o masina clasica ar avea o viteza de 1/5x MIPS (milion instruction per second). Viteza de prelucrare pe pipeline este 1/x MIPS.

Folosirea pipeline determina un compromis între latenta (latency – cât dureaza executia unei instructiuni) si largirea de banda a procesului. (bandwidth – câte MIPS are UCP). Pentru un ciclu de ceas T μs si n segmente, latenta este nT si largimea de banda -1/T MIPS.

Arhitecturi superscalare

Date fiind avantajele benzii de asamblare, ar fi de dorit mai multe din acestea. În figura 2.4 este prezentata o posibila proiectare a unui UCP în banda de ansamblare duala. Pentru a putea lucra în paralel, cele 2 instructiuni nu trebuie sa-si dispute resursele (de exemplu registrele) si nici una nu trebuie sa depinda de rezultatul celeilalte. Fie compilatorul trebuie sa garanteze ca ipoteza anterioara e respectata, fie conflictele sunt detectate si eliminate pe parcursul executiei, cu ajutorul unui hardware suplimentar.

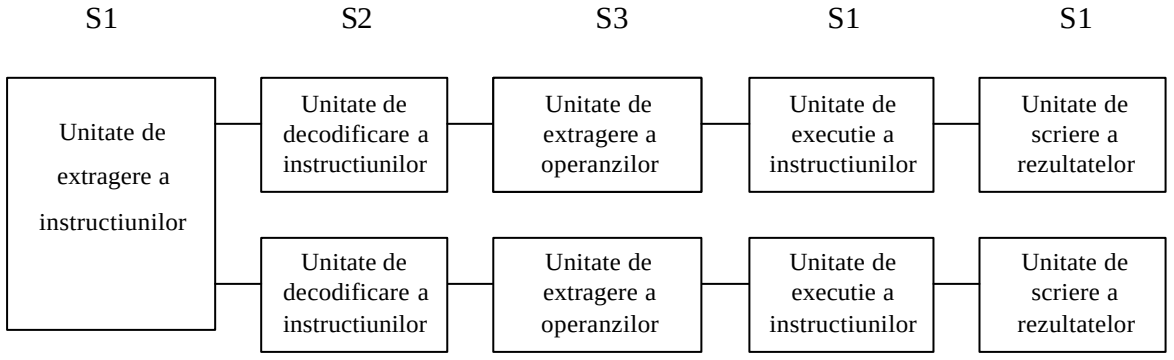

Fig. 2.4. Banda de ansamblare duala cu 5 segmente

Cu toate ca benzile de ansamblare, simple sau duale, sunt folosite pe masinile RISC (386 si predecesorii nu aveau nici una), începând cu 486, Intel a introdus o banda de ansamblare în procesoarele sale.

Pentium are doua benzi de ansamblare asemanatoare cu cele din fig. 2.4, desi împartirea între segmentele 2 si 3 (numite decode-1 si decode-2) este putin diferta fata de cea din exemplul nostru. Banda de asamblare principala, numita u pipeline, poate executa orice instructiune Pentium, în timp ce a doua banda, numita v pipeline, poate executa doar instructiuni pentru întregi si o instructiune simpla în virgula mobila – FXCH.

Reguli destul de complexe determina daca instructiunile sunt compatibile, astfel încât sa poata fi executate în paralel. Daca instructiunile sunt incompatibile, doar prima este executata (pe banda u), iar a doua este pastrata si împerecheata cu o instructiune care o va urma. Instructiunile sunt executate în ordine. Din aceasta cauza compilatoarele specifice Pentium, care produceau perechi compatibile de instructiuni, puteau sa produca programe mai rapide. Pentru programe cu numere întregi, Pentium cu cod optimizat pentru el este de 2 ori mai rapid decât 486 la aceeasi frecventa de ceas. Acest câstig se datoreaza celei de-a 2-a benzi de asamblare.

Alte UCP utilizeaza abordari cu totul diferite. Idea de baza este de a avea o singura banda de ansamblare, dar cu mai multe unitati functionale, asa cum se observa în figura 2.5

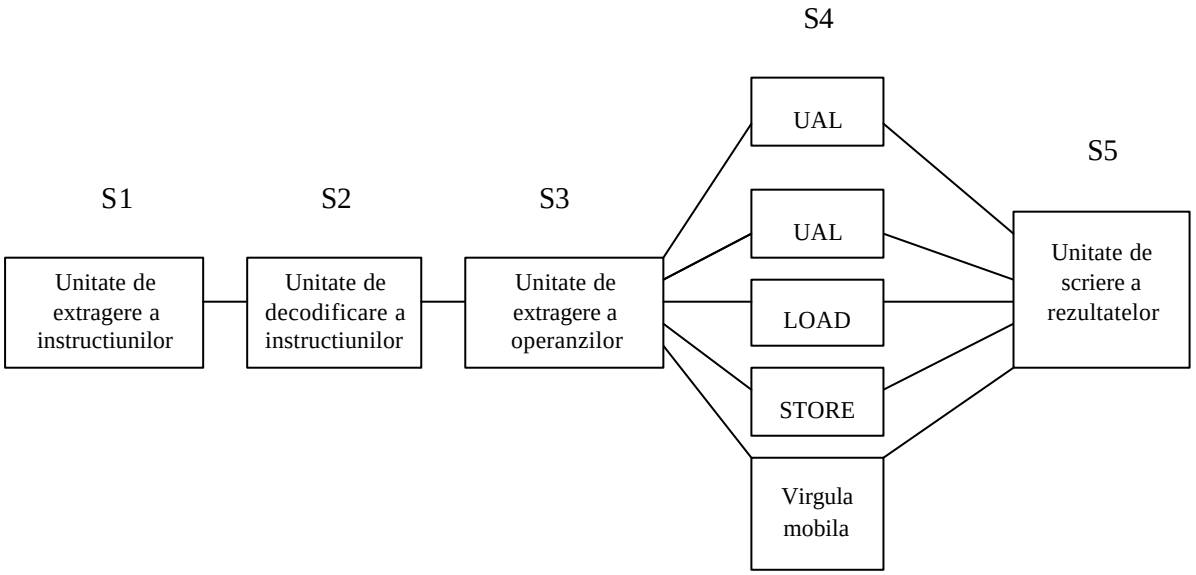

Fig. 2.5. Un processor superscalar cu 5 unitati functionale

Procesorul Pentium II are o structura asemanatoare cu cea din fig. 2.5. Pentru aceasta abordare a fost introdus termenul arhitectura superscalara (Agerwala si Cocke, 1987). Radacinile lui sunt vechi de 30 de ani, în calculatorul CDC6600, care extragea o instructiune la fiecare 100nsec si o trimitea pentru executie în paralel, uneia din cele 10 unitati functionale, în timp ce UCP se ocupa cu extragerea instructiunei urmatoare.

Ar rezulta din figura ca segmental S3 poate lansa instructiunile mult mai repede decât le poate executa segmental S4. În realitate, majoritatea unitatilor functionale din segmental S4 au nevoie de mai mult timp decât un ciclu de ceas pentru a-si face treaba, în mod cert cele care acceseaza memoria sau care lucreaza în virgula mobila.

Dupa cum se poate observa pot exista mai multe UAL în segmentul S4.

2.6. Paralelism la nivelul procesului

Cresterea vitezei procesoarelor este o problema deschisa. Paralelismul la nivel de instructiune ajuta putin, iar banda de ansamblare si operarea superscalara cresc performantele de 5-10 ori. Singura modalitate de a creste mai mult viteza este de a proiecta calculatoare cu mai multe UCP.

Calculatoare matriceale

Un numar mare de probleme de fizica si inginerie implica matrici, vectori sau macar o structura regulata. Organizarea regulata si structura unor astfel de programe le fac potrivite pentru executia paralela.

Un calculator matricial (array processor) aste alcatuit dintr-un numar mare de procesoare identice ce executa aceeasi secventa de instructiuni pe seturi de date diferite. Primul calculator vectorial din lume a fost calculatorul ILLIAC IV al Universitatii din Illinois, prezentat în Fig. 2.6.

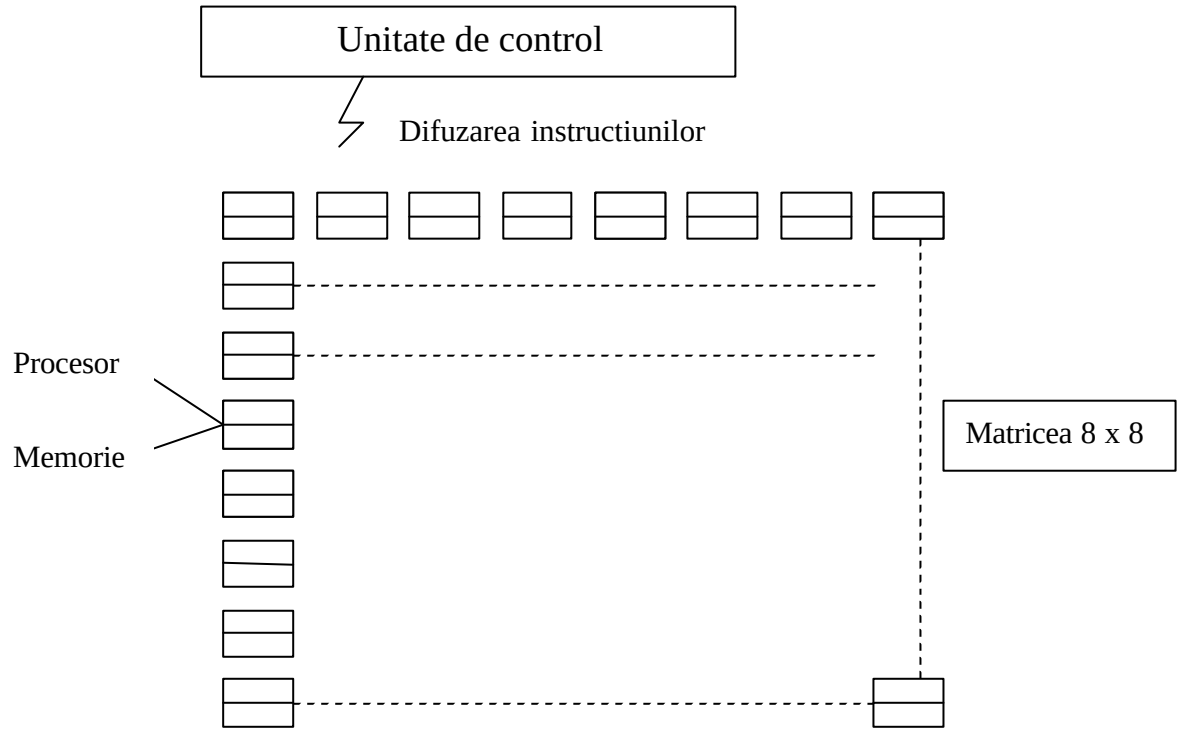

Fig. 2.6. Calculator matricial de tipul ILLIAC IV

Planul initial prevedea construirea unei masini din 4 cadrane, fiecare cadran având o matrice de 8 X 8 procesoare + memorie. O singura unitate de control pentru un cadran difuza instructiunile, care erau executate sincron pe toate procesoarele. A fost construit un singur cadran si acesta a obtinut o performanta de 50 Mflops (mega flotant operations per second). Un processor vectorial seamana foarte mult cu unul matricial. Dar spre deosebire de calculatorul matricial, toate operatiile de adunare sunt executate de un singur sumator, folosind intens benzile de ansamblre. Compania fondata de Segmour Cray, Cray Research, a produs mai multe calculatoare vectoriale, începând cu 1974. Acestea mai erau numite supercalculatoare. Cray Research face parte în prezent din SGI.

Atât calculatoaele matriciale cât si cele vectoriale pot executa operatii unice (de exemplu: aduna element cu element 2 vectori). În timp ce calculatorul matricial foloseste câte un sumator pentru fiecare element al vectorilor, calculatorul vectorial dispune de conceptul registru vectorial, care este alcatuit dintr-un set de registre conventionale ce pot fi încarcate din memorie cu o singura instructiune, pe care de fapt le încarca serial din memorie. Pe urma, o instructiune de adunare vectoriala executa adunarea element cu element a celor e vectori, trimitându-i din cele 2 registre vectoriale catre sumatorul cu banda de asamblare. Rezultatul la iesirea sumatorului este tot un vector, ce poate fi memorat într-un registru vectorial, fie folosit drept operand pentru o alta operatie.

Înca se mai produc calculatoare matriceale, dar segmental de piata ocupat de acestea este în scadere. Calculatoarele matriceale executa anumite operatii mai bine decât cele vectoriale, deci au nevoie de mult mai mult hardware si soft dificil de programat. Pe de alta parte, procesoarele vectoriale pot fi adaugate unui processor conventional.

Multiprocesoare

Elemente de prelucrare dintr-un process matriceal nu sunt UCP-uri independente, deoarece partajeaza o singura unitate de control. Multiprocesorul este un sistem paralel cu mai multe UCP-uri complet folosite. Aceasta partajeaza o memorie comuna de memorie. De vreme ce fiecare UCP poate citi sau scrie orice locatie de memorie, ele trebuie sa se coordoneze prin software astfel încât sa evite conflictele.

Sunt posibile diverse scheme de implementare. Cea mai simpla este cu mai multe UCP si o singura memorie, ca în fig. 2.7.a.

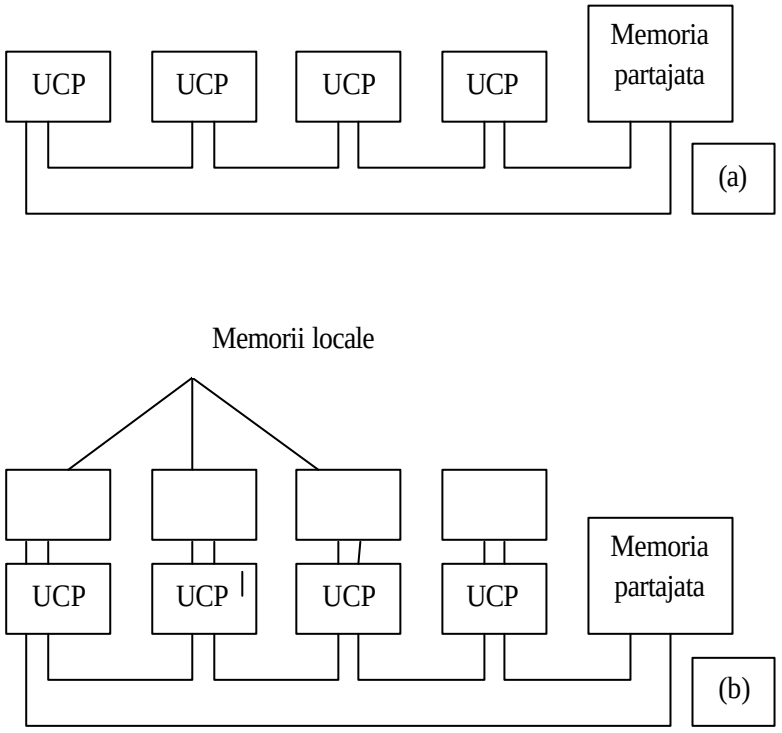

Fig. 2.7. (a) Un multiprocessor cu o magistrala
(b) Un multicaculator cu memorii locale

Ne putem da seama ca vor apare conflicte daca un numar mare de procesoare rapide vor încerca permanent sa acceseze memo ria folosind aceasi magistrala. Pentru a rezolva problema au fost imaginate diverse scheme. O varianta, cea din figura 2.7.b asigura fiecarui processor o memorie locala. Aceasta memorie poate fi folosita pentru codul programului si acele date care nu trebuie protejate. Accesul la memoria propie nu implica folosirea magistralei principale, reducând mult traficul pe magistrala. Sunt posibile si alte scheme – de exemplu cu memorii intermediare.

Avantajul multiprocesoarelor fata de alte categorii de calculatoare este acela ca modelul de programare a unei singure memorii partajate este usor de folosit.

Multicalculatoarele

Desi multicalculatoarele cu un numar mic de procesoare sunt usor de construit (≤64), cele mai mari sunt surprinzator de greu de construit. Dificultatea consta în conectarea tuturor procesoarelor la memorie. Pentru a evita aceste probleme, multi proiectanti au renuntat la ideia unei memorii partajete si au construit pur si simplu sisteme alcatuite dintr-un numar mare de calculatoare fiecare având memorie proprie, dar fara o memorie comuna. Aceste sisteme sunt numite multicalculatoare (multicomputers). UCP-urile dintr-un multicalculator comunica prin mesaje de mare viteza.

De exemplu, la transputere, comunicatia se face legaturi INMOS seriale de ma re viteza. Pentru sisteme mari, conectarea fiecarui calculator cu toate celelalte nu este o idee practica, astfel ca se folosesc topologii de tip matrice 2D si 3D, arbori sau inele. Rezulta ca mesajele de la un calculator vor trebui sa treaca de mai multe ori printr-unul sau mai multe calculatoare sau comutatoare intermediare pentru a ajunge la destinatie. Pe de alta parte timpii de transmitere de ordinul câtorva microsecunde pot fi obtinuti. Multicalculatoare cu 10.000 UCP s-au construit deja si sunt operationale.

Pentru ca multiprocesoarele sunt usor de construit se cerceteaza proiectare unor sisteme hibride care sa îmbine avantajele ambelor.